Part 2 version 2. More Python, Variables, Math

Notes, examples, and the assignment at : https://code.actor

Variables

A “variable” is a place in computer memory where your program can store information while the program is running. There are several data “types” but that’s a whole other discussion.

In Python you declare a variable by making up a name and assigning a value to it. The value of a variable can change during execution, that is the value of a variable can “vary.” That’s why they are called variables. Picking good names is one of the hardest problems.

Name rules: Lowercase a-z, uppercase A-Z, 0-9 numbers, and the underscore _, can’t be a Python reserved word, can’t start with a number.

```
myDogName = “Rover”     # this is a camelCase style name
my_dog_name = “Rover”   # this is a snake_case style name
myAge = 12              # the = is called the assignment operator
my_age = 12
speed = 34
number_1 = 3
number_2 = 4
```

Math

There are several “arithmetic operators” for doing math operations. In these examples, variable var_3 is assigned values like :

```
var_3 = var_1 + value_2 # adds var_1 and var_2
var_3 = var_1 - 5       # subtracts 5 from var_1
var_3 = 56 * var_2      # multiplies 56 by var_2
var_3 = var_1 / 10      # divides var_1 by 10
var_3 = var_1 % var_2   # divides var_1 by var_10, returns remainder
var_3 = var_1 ** var_2  # var_1 to the power of var_2
```

Math operations require both “operands” to be numeric.

```
var_1 = 10        # whole numbers are sometimes called “integers.”
var_2 = 3.314159  # decimals are AKA “floating point numbers.”
var_3 = var_1 + var_2    # var_3 becomes  13.14159
```

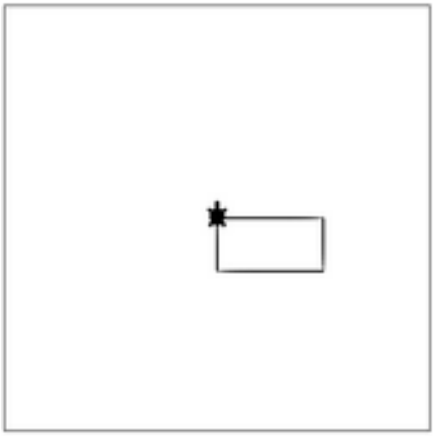
If one of the “operands” is a “string,” Python can’t do math with it. Python (and most other languages) will throw an error. If both operands are strings, the + operator will “*concatenate”* the strings.

```
var_1 = "taco"
var_2 = "cat"
var_3 = var_1 + var_2   # var_3 becomes “tacocat”
```

Part 2 version 2, Page2. Python, Variables, Math

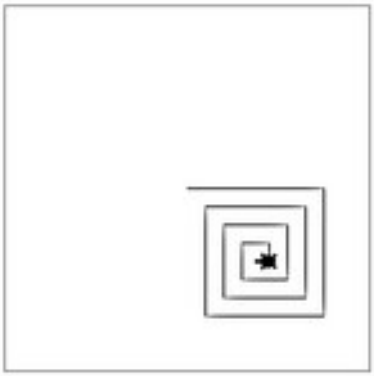
Suppose you want rectangles that are automatically half as tall as they are wide. No matter what you set **`rectangle_width`** to, **`rectangle_height`** will *automagically* be half as big.

```
rectangle_width = 100

rectangle_height = rectangle_width / 2

forward(rectangle_width)
right(90)
forward(rectangle_height)
right(90)
forward(rectangle_width)
right(90)
forward(rectangle_height)
```



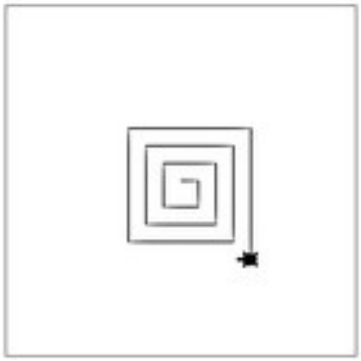
You haven't learned about *loops* yet, but just take my word for it that the block of code after "`for i in range...`" repeats 10 times ...

```
line_length = 150
how_many_loops = 14

for i in range(0,how_many_loops):
    forward(line_length)
    right(90)
    line_length = line_length - 10
```



Pay special attention to **`line_length = line_length - 10`**
Each time the loop... uh... loops... (we say "iterates"), line_length will become 10 smaller.

```
line_length = 20
how_many_loops = 14

for i in range(0,how_many_loops):
    forward(line_length)
    right(90)
    line_length = line_length + 10
```



... Or here, we start small and the lines get bigger on each "iteration." We will learn about loops later, but notice the use of "arithmetic operators" to change the value of variables.